UE21CS352B - Object Oriented Analysis & Design using Java

Mini project report

project management system

Team Details

Nandish N S : PES1UG21CS364
Nandan N Prabhu : PES1UG21CS362
Manoj Kumar R : PES1UG21CS330
Sai Kashyap : PES1UG21CS349

6th Semester F Section

Prof. Bhargavi Mokashi

Assistant Professor

January - May 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013) 100ft Ring Road, Bengaluru – 560 085, Karnataka, India

Problem statement

The project management system allows administrators to create and manage projects, assign developers to projects, view work reports, and collect feedback. Developers can report their work hours, view assigned projects, and provide feedback to the admin. The system follows a modular architecture with separate components responsible for different aspects of user and project management, promoting maintainability and scalability.
Key features:

- Admin can create projects
- Admin can assign developers to the project
- Developers can report their work
- Admin can view the reported work by developers
- Developers can give feedback to the admin

Classes Used :

- Work
- User
- Project
- Developer
- Feedback

Controllers Used :

- UserController

- ProjectController

- DeveloperController

- FeedbackController

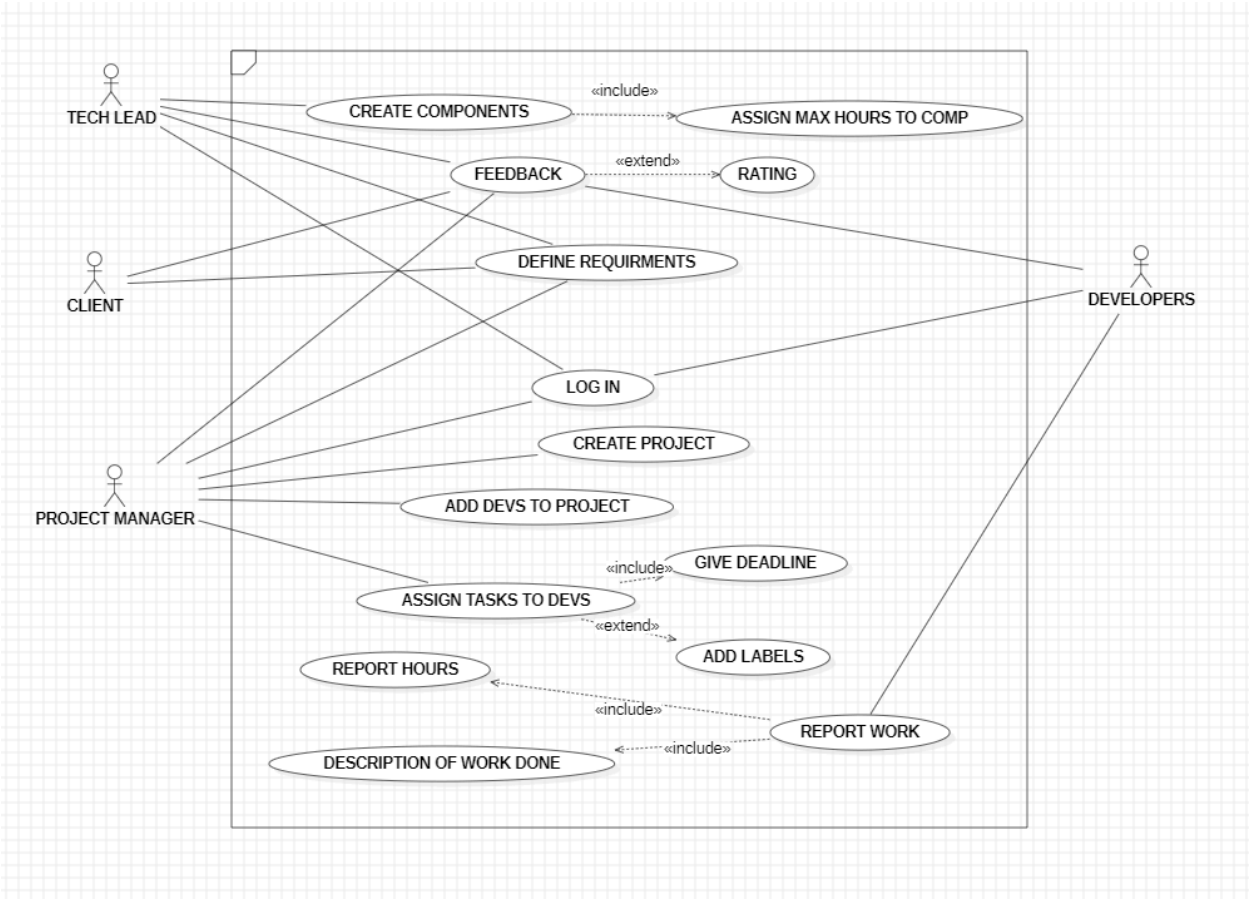
- WorkController

Interfaces Used :

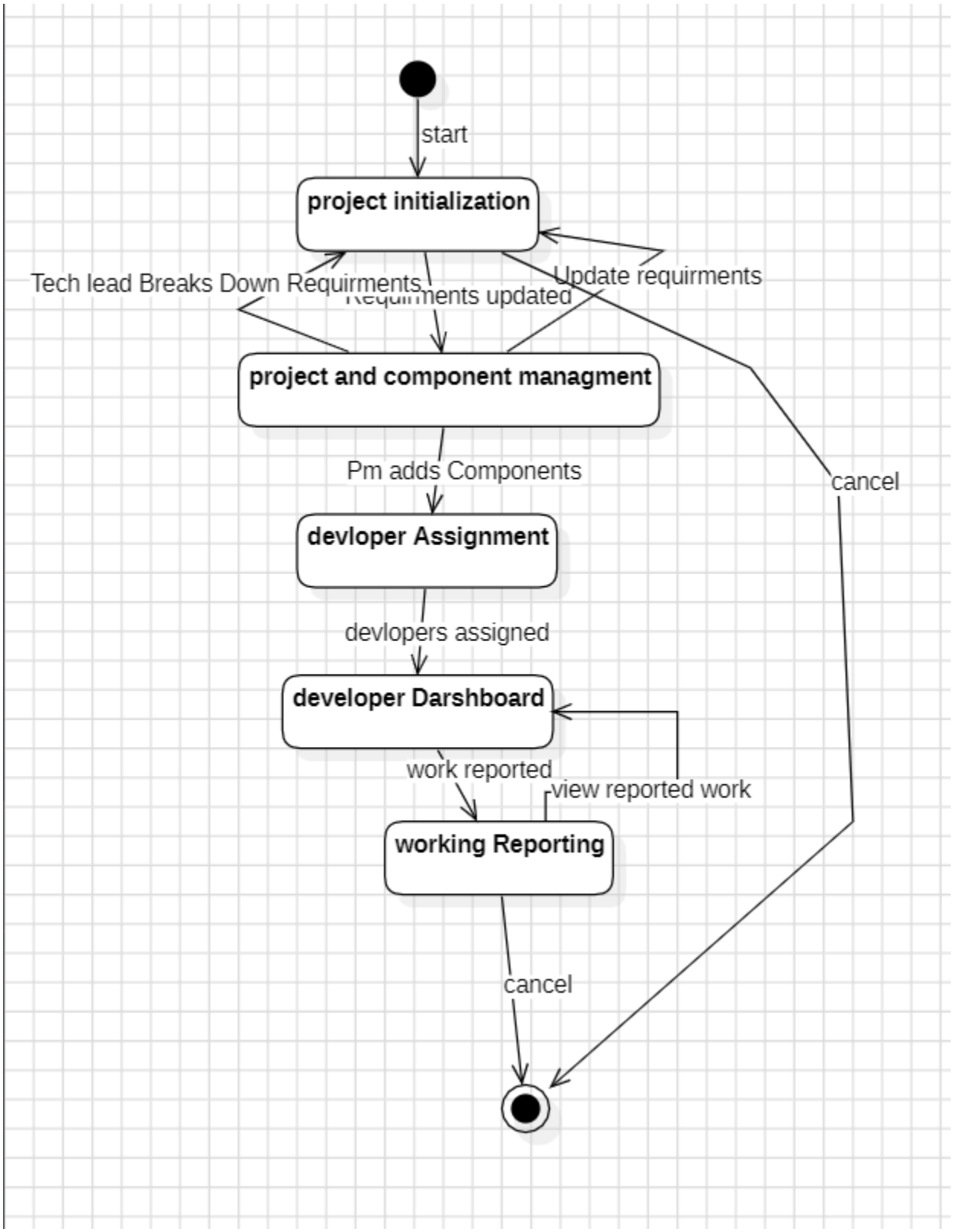
- RoleCommand

Models

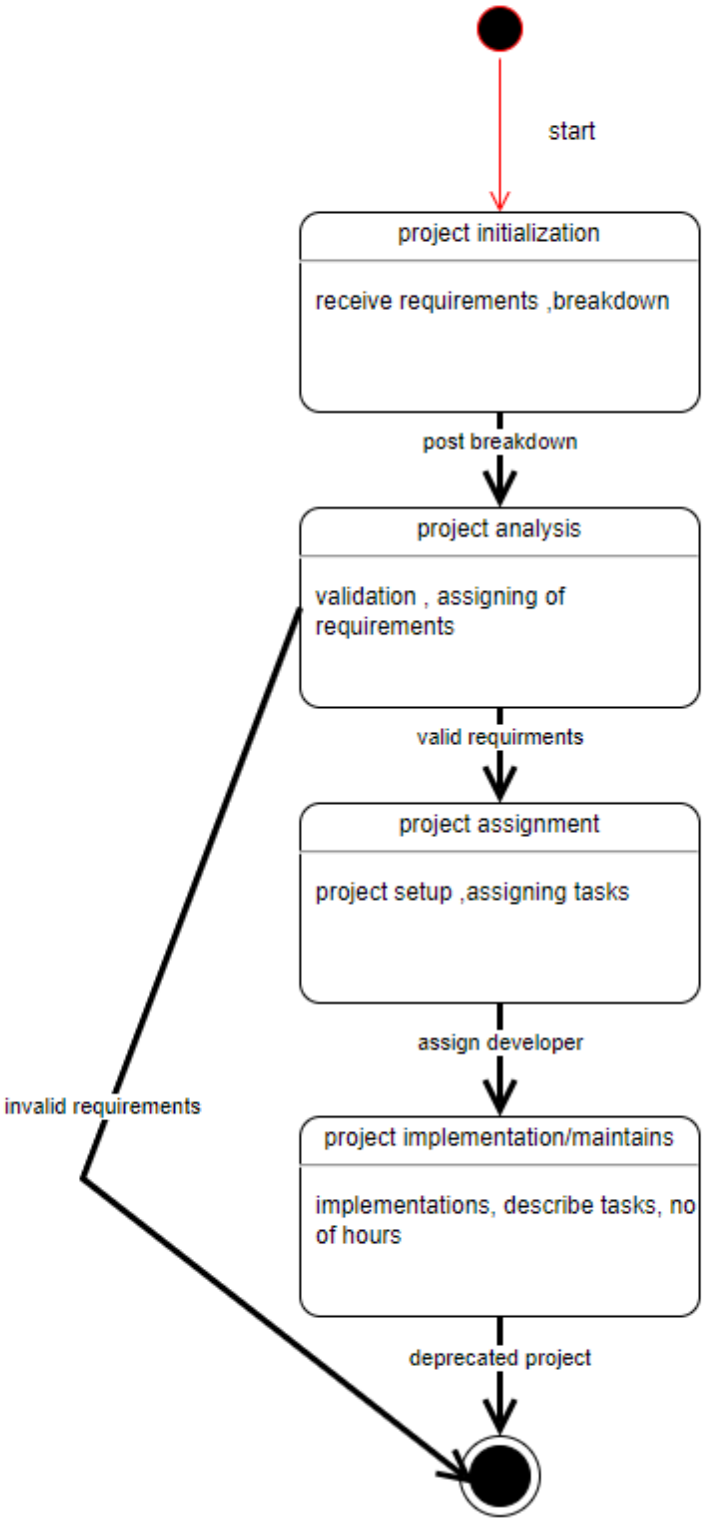
USE CASE DIAGRAM



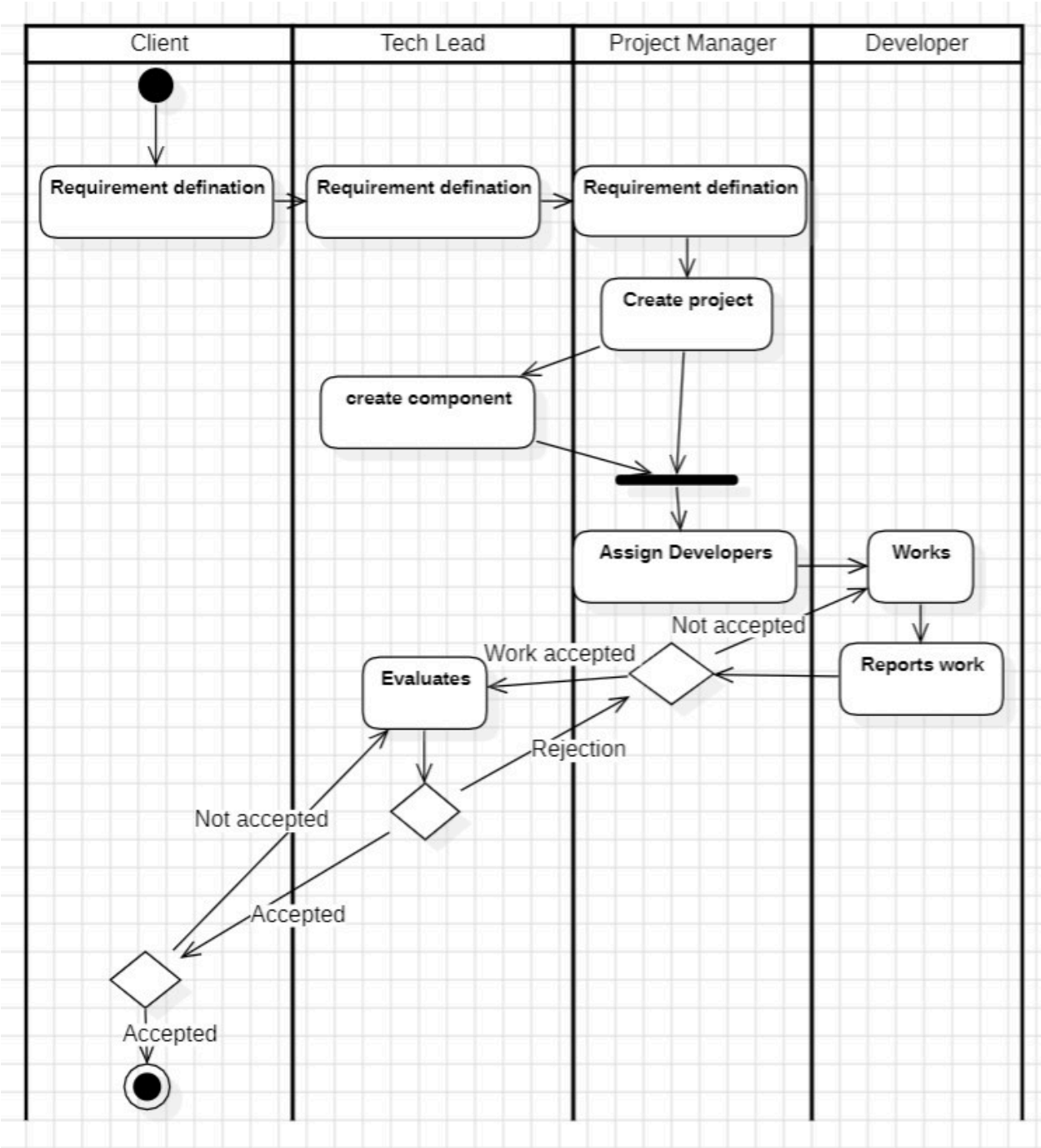
State diagram



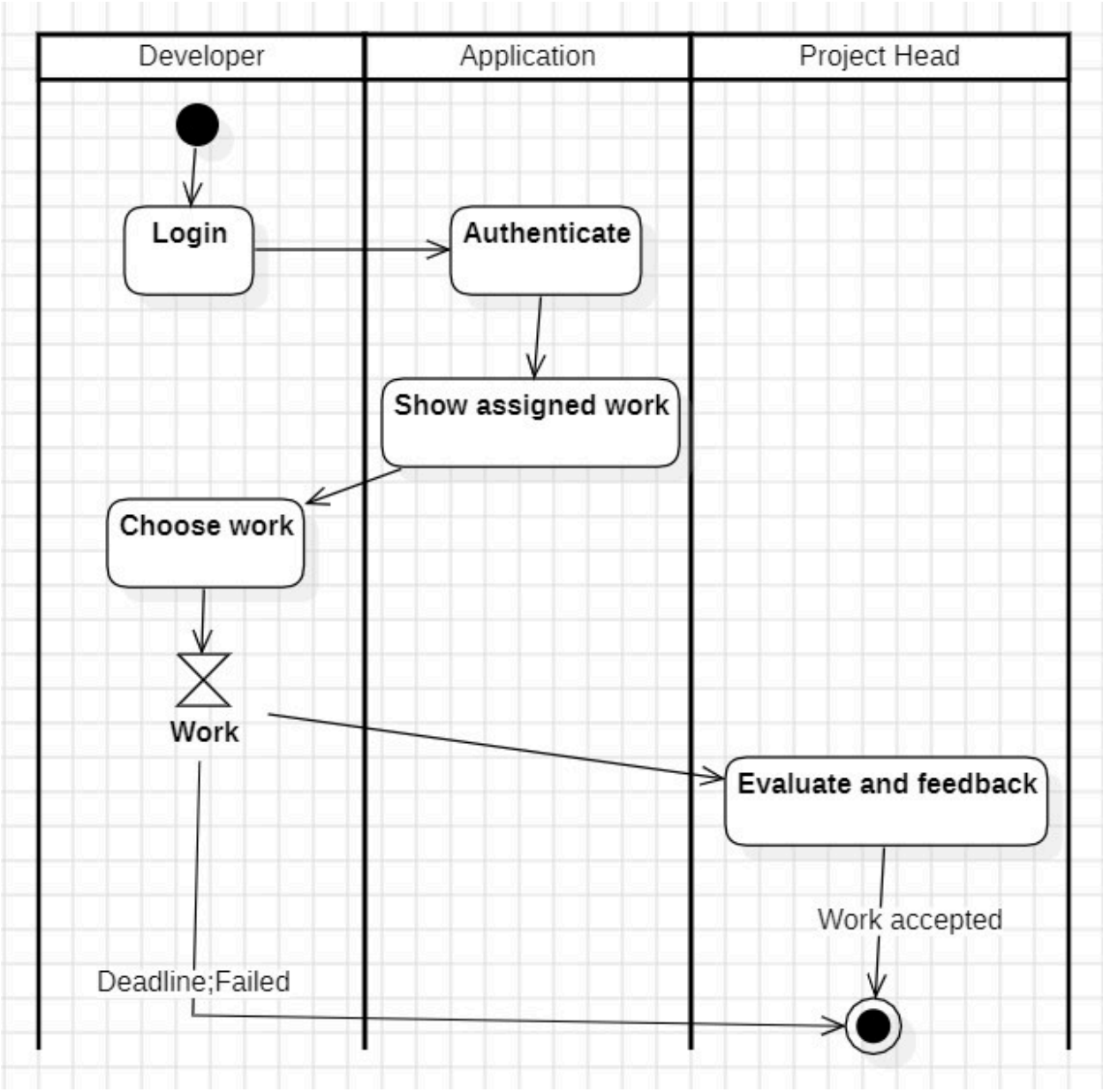
V2:


start
project initialization
receive requirements ,breakdown
post breakdown
project analysis
validation , assigning of requirements
valid requirments
project assignment
project setup ,assigning tasks
assign developer
invalid requirements
project implementation/maintains
implementations, describe tasks, no of hours
deprecated project

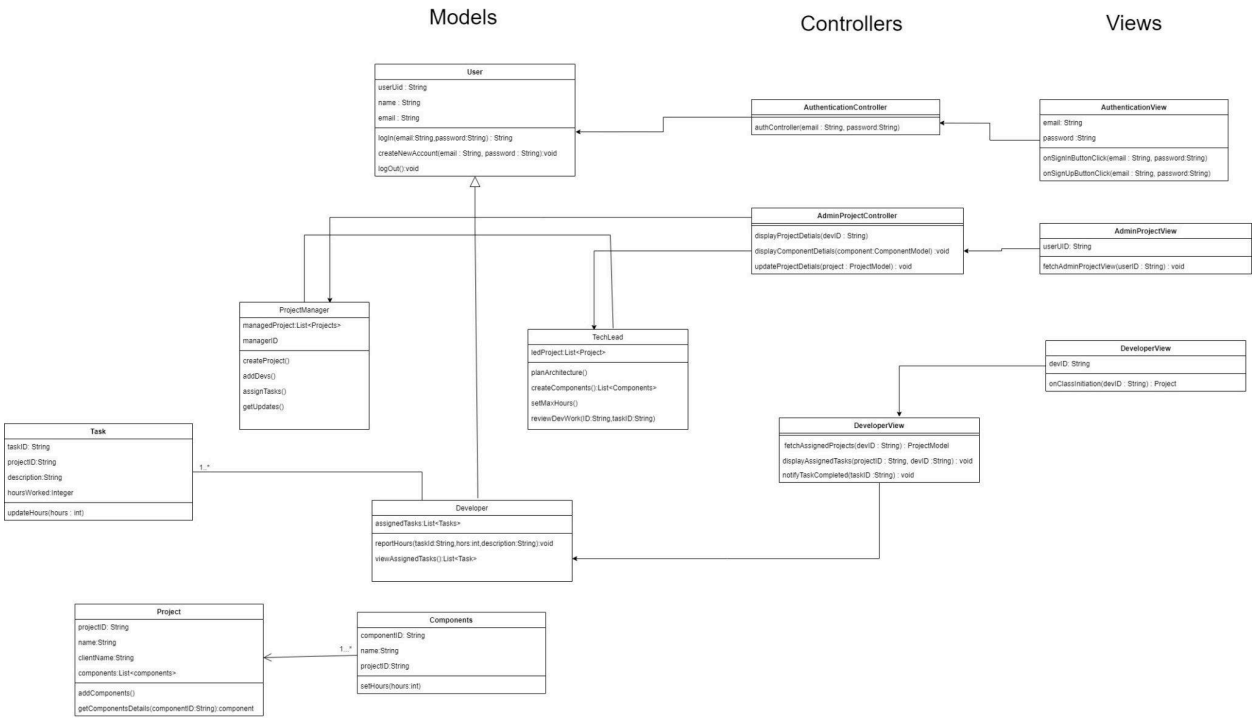
Activity Diagram 1 :



Activity Diagram 2 :



Class Diagram



Design principles:

Single Responsibility Principle (SRP):

Each class or component has a single responsibility. The UserController, ProjectController, DeveloperController, and FeedbackController each handle specific aspects of user management, project management, work reporting, and feedback handling, respectively.

Open/Closed Principle (OCP):

The code structure allows for extension without modification. For instance, new functionalities can be added to the system by introducing new controllers, services, or helper classes without altering existing code.

Dependency Inversion Principle (DIP):

Dependency Injection is employed throughout the codebase, allowing classes to depend on abstractions (such as interfaces) rather than concrete implementations. This facilitates loose coupling and promotes easier testing and flexibility in component dependencies.

Design patterns:

Proxy pattern : ProtectionProxy

Usage in the Code: The ProtectionProxy class acts as a proxy for checking permissions before allowing the deactivation of a project in the ProjectController.
The Proxy pattern provides a surrogate or placeholder for another object to control access to it. In this case, the ProtectionProxy controls access to the ProjectManager by checking permissions before forwarding requests. This helps in implementing cross-cutting concerns, such as security checks, without modifying the core functionality of the ProjectController.

Builder: feedback
The Feedback class includes a nested feedback_builder static class, which implements the Builder pattern. This builder class provides a convenient way to construct Feedback objects with different configurations, allowing clients to set feedback and star attributes using setter methods (setFeedback, setStar) before calling the build method to create the Feedback instance

Command : AdminRoleCommand
The AdminRoleCommand class implements the RoleCommand interface, defining the userCreation method. This method represents a specific command that can be executed by invoking the userCreation method.
The AdminRoleCommand class encapsulates the command for creating users with admin roles. The Command pattern encapsulates a request as an object, allowing parameterization of clients with queues, requests, and operations. In this scenario, the AdminRoleCommand encapsulates the user creation operation, providing a way to parameterize the operation with user-specific data and execute it independently. This promotes loose coupling between the invoker (controller) and the receiver (user creation logic).

Strategy : WorkController

The `Work` class in the project encapsulates various operations related to work management within a project. It includes methods for initializing a connection to a Firestore database, retrieving and updating work details, and calculating pay details for a specific user. The `Work` class utilizes the Strategy pattern, employing `OperationMultiply` for performing a multiplication operation within the `getPayDetails` method. The associated `WorkController` handles incoming API requests related to work, invoking appropriate methods in the `Work` class to process and respond to client requests, including fetching reported work and pay details. This design promotes loose coupling between the controller and the `Work` class, allowing for more modular and maintainable code.

Individual contributions

- **Nandish N S**
    - frontend
    - MVC
    - strategy pattern

- **Nandan N Prabhu**
    - Builder pattern
    - Frontend

- **Manoj Kumar R**
    - Factory pattern
    - Command pattern

- **Sai Kashyap**
    - Proxy pattern